Mini Max

Morris Game Using Mini Max algorithm:

Assuming the Mini Max algorithm is being used for White Pieces, an input value X indicates potential positions for White while still adhering to the game's rules. By utilizing the Mini Max algorithm, all feasible moves for White can be generated from the provided input string.

For given depth value, the algorithm goes on creating all the outputs at that depth. When the depth value matches the input depth, a static estimation evaluates the node estimation with the assumption that White is at the MAX level. This evaluation produces the most favorable board position for White. Overall, the Mini Max algorithm calculates all potential board positions for a given input and depth. If the algorithm is playing as White, it selects the best move for White based on this evaluation.

Alpha Beta Pruning

Morris Game Using Alpha Beta Pruning algorithm:

The Alpha-Beta Pruning algorithm functions similarly to Mini-Max, but it omits evaluating nodes that do not influence the ultimate outcome in order to determine the optimal move for White pieces.

Morris Game Opening:

This is the beginning of the Morris Game, in which players place their pieces on the board and aim to create a Mill.

Morris Game Mid game - End game:

Players are permitted to move their pieces by one position, except when they have only three pieces remaining on the board.

Two cases in which alpha-beta produces savings over MINIMAX

- **1st case**

C:\Users\alekhya\NineMenMorris>python MiniMaxGame.py board7.txt MiniMaxGame1.txt 4

Board Position: WxxxWxWxWxxxBBWWBBBWBx

Positions evaluated by static estimation: 6188

MINIMAX estimate: 1987

C:\Users\alekhya\NineMenMorris>python ABGame.py board7.txt AlphaBetaGame1.txt 4

Input Board Position: WxxWWxWxxxxxBBWWBBBWBx

Depth: 4

Board Position: WxxxWxWxWxxxBBWWBBBWBx

Positions evaluated by static estimation: 496

ALPHA-BETA estimate: 1987

- **2nd Case**

C:\Users\alekhya\NineMenMorris>python MiniMaxGame.py board8.txt MiniMaxGame2.txt 4

Board Position: WBBxxxxBxxxBWxxxxWxxWx

Positions evaluated by static estimation: 19244

MINIMAX estimate: -1008

C:\Users\alekhya\NineMenMorris>python ABGame.py board8.txt AlphaBetaGame2.txt 4

Input Board Position: WBBxxxxBxxxBWxxxxxWxWx

Depth: 4

Board Position: WBBxxxxBxxxBWxxxxWxxWx

Positions evaluated by static estimation: 4374

ALPHA-BETA estimate: -1008

two examples where your static evaluation function produces different moves than the standard evaluation function

- ### 1st Example

C:\Users\alekhya\NineMenMorris>python MiniMaxGame.py board6.txt MinimaxGameNormal2.txt 3

Board Position: WxxWWxWxxxxxBBWWBBBWBx

Positions evaluated by static estimation: 1696

MINIMAX estimate: 991

C:\Users\alekhya\NineMenMorris>python MiniMaxGameImproved.py board6.txt MinimaxGameImproved2.txt 3

Input Board Position: WxBWWxxxxWxxBBWWBBBWBx

Depth: 3

Board Position: WxBWWxxxxWWxBBWxBBBWBx

Positions evaluated by static estimation: 1696

MINIMAX estimate: 22

- ### 2nd Example

C:\Users\alekhya\NineMenMorris>python MiniMaxGame.py board3.txt MinimaxGameNormal1.txt 3

Board Position: WWxxxxxWWWWBBBBWxxxBxB

Positions evaluated by static estimation: 5037

MINIMAX estimate: 987

C:\Users\alekhya\NineMenMorris>python MiniMaxGameImproved.py board3.txt MinimaxGameImproved1.txt 3

Input Board Position: WWxxxxxWWWWBBBBxxBWBxB

Depth: 3

Board Position: WWxxxxWxWWWBBBBxxBWBxB

Positions evaluated by static estimation: 5037

MINIMAX estimate: 10

Improved Static Estimation:

The Handout technique utilizes the count of Black and White pieces to determine the node/position value. The Improved Static Estimation method also takes into account the potential Mill positions for a given piece in the evaluation process and Also results in the most advantage position And evaluates the number of white pieces that are defending the mill position for black If the game is being played for White on a particular board, the Improved Static Estimation method additionally factors in the number of future potential Mill formations after executing a particular move.

Conclusions:

The Mini Max algorithm evaluates all feasible node values, including those that do not influence the ultimate outcome.

Alpha Beta Pruning is an optimized version of the Mini Max algorithm that decreases the number of node evaluations required.

Although the outcomes obtained using Mini Max and Alpha Beta Pruning are similar, Alpha Beta Pruning reduces the number of node evaluations that must be conducted.